RUBY SOCKET PROGRAMMING

Ruby provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.

Ruby also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

This tutorial gives you understanding on most famous concept in Networking - Socket Programming

What is Sockets?

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The *socket* library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Sockets have their own vocabulary:

Term	Description
domain	The family of protocols that will be used as the transport mechanism. These values are constants such as PF_INET, PF_UNIX, PF_X25, and so on.
type	The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
protocol	Typically zero, this may be used to identify a variant of a protocol within a domain and type.
hostname	The identifier of a network interface: • A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation • A string "<broadcast>", which specifies an INADDR_BROADCAST address. • A zero-length string, which specifies INADDR_ANY, or • An Integer, interpreted as a binary address in host byte order.
port	Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

A Simple Client:

Here we will write a very simple client program which will open a connection to a given port and given host. Ruby class **TCPSocket** provides *open* function to open such a socket.

The **TCPSocket.open(hosname, port)** opens a TCP connection to *hostname* on the *port*.

Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits:

```
require 'socket'      # Sockets are in standard library

hostname = 'localhost'
port = 2000

s = TCPSocket.open(host, port)

while line = s.gets   # Read lines from the socket
  puts line.chop      # And print with platform line terminator
end
s.close               # Close the socket when done
```

A Simple Server:

To write Internet servers, we use the **TCPServer** class. A TCPServer object is a factory for TCPSocket objects.

Now call **TCPServer.open(hostname, port** function to specify a *port* for your service and create a **TCPServer** object.

Next, call the *accept* method of the returned TCPServer object. This method waits until a client connects to the port you specified, and then returns a *TCPSocket* object that represents the connection to that client.

```
require 'socket'               # Get sockets from stdlib

server = TCPServer.open(2000)  # Socket to listen on port 2000
loop {                         # Servers run forever
  client = server.accept       # Wait for a client to connect
  client.puts(Time.now.ctime)  # Send the time to the client
  client.puts "Closing the connection. Bye!"
  client.close                 # Disconnect from the client
}
```

Now run this server in background and then run above client to see the result.

Multi-Client TCP Servers:

Most servers on the Internet are designed to deal with large numbers of clients at any one time.

Ruby's *Thread* class makes it easy to create a multithreaded server.one that accepts requests and immediately creates a new thread of execution to process the connection while allowing the main program to await more connections:

```
require 'socket'                # Get sockets from stdlib

server = TCPServer.open(2000)   # Socket to listen on port 2000
loop {                          # Servers run forever
  Thread.start(server.accept) do |client|
    client.puts(Time.now.ctime) # Send the time to the client
 client.puts "Closing the connection. Bye!"
    client.close                # Disconnect from the client
  end
}
```

In this example you have a permanent loop, and when server.accept responds, a new thread is created and started immediately to handle the connection that has just been accepted, using the connection object passed into the thread. However, the main program immediately loops back and awaits new connections.

Using Ruby threads in this way means the code is portable and will run in the same way on Linux, OS X, and Windows.

A Tiny Web Browser:

We can use the socket library to implement any Internet protocol. Here, for example, is code to fetch the content of a web page:

```
require 'socket'

host = 'www.tutorialspoint.com'     # The web server
port = 80                           # Default HTTP port
path = "/index.htm"                 # The file we want

# This is the HTTP request we send to fetch a file
request = "GET #{path} HTTP/1.0\r\n\r\n"

socket = TCPSocket.open(host,port)  # Connect to server
socket.print(request)               # Send request
response = socket.read              # Read complete response
# Split response at first blank line into headers and body
headers,body = response.split("\r\n\r\n", 2)
print body                          # And display it
```

To implement the similar web client you can use a prebuilt library like **Net::HTTP** for working with HTTP. Here is code that does the equivalent of the previous code:

```
require 'net/http'                  # The library we need
host = 'www.tutorialspoint.com'     # The web server
path = '/index.htm'                 # The file we want

http = Net::HTTP.new(host)          # Create a connection
headers, body = http.get(path)      # Request the file
if headers.code == "200"            # Check the status code
  print body
else
  puts "#{headers.code} #{headers.message}"
end
```

Please check similar libraries to work with FTP, SMTP, POP, and IMAP protocols.

Further Readings:

I have given you a quick start with Socket Programming. It's a big subject so its recommended to go through the following link to find more detail on Ruby Socket Library and Class Methods.